Technische Hochschule Mittelhessen

Prof. Dr.-Ing. A. Christidis

**Fachbereich Mathematik,
Naturwissenschaften und Informatik**

7. Februar 2012

Klausur
Computergrafik
für Bachelor-Studierende
WS 2011 / 12

– Lösungshilfe –

Personalien:

Name, Vorname: .....

Matrikelnummer: .....

Hinweise:

- **Die Bearbeitungszeit beträgt 90 Minuten.**
- **Alle schriftlichen Hilfsmittel sind zugelassen; andere Hilfsmittel, insb. elektr. Rechen- und Kommunikationsapparate, dürfen nicht verwendet werden.**
- **Ausgesprochene Folgefehler (durch Übertragung falscher Zwischenergebnisse) werden in Folgerechnungen als richtig gewertet.**
- **Die Aufgaben sollen nur auf diesen Blättern (inkl. Rückseite) bearbeitet werden. Bei Bedarf wird zusätzliches Papier zur Verfügung gestellt.**
- **Zur sicheren Zuordnung aller Lösungen wird um eine persönliche Kennung (Name u./o. Matrikelnr.) auf allen Blättern gebeten.**
- **Auf Wunsch darf auch Bleistift verwendet werden.**

Zur leichteren Lesbarkeit werden Substantive nur in einem Geschlecht („Nutzerin“) verwendet.

1. **Aufgabe** (10 Punkte)

a) Sie lernen jemanden kennen, der seit Jahrzehnten als Entwickler auf dem Gebiet der Informationstechnologie tätig ist. Sie erfahren, daß er ein Verfahren entwickelt hat, mit dem Paßbilder nahezu überflüssig werden: Auf der Grundlage weniger signifikanter Punkte, die im Paß als Raum-Koordinaten gespeichert wären, wird die von ihm erstellte Software eine virtuelle Skulptur erstellen, die (Experten zufolge) weit aussagekräftiger als ein Foto sein wird.

Handelt es sich bei dieser wichtigen Person um einen Experten für Bildbearbeitung, für Bildverarbeitung, für Computergrafik, für alle drei oder für keine von ihnen? Bitte kreuzen Sie die richtige(n) Antwort(en) an!

	Experte für Bildbearbeitung
	Experte für Bildverarbeitung
X	Experte für Computergrafik
	Experte für keines der vorgenannten Gebiete

b) Jemand zeigt Ihnen zwei Bilder von seiner neuen Implementierung des Linienalgorithmus von Bresenham (Abb. 1.1, Abb. 1.2).

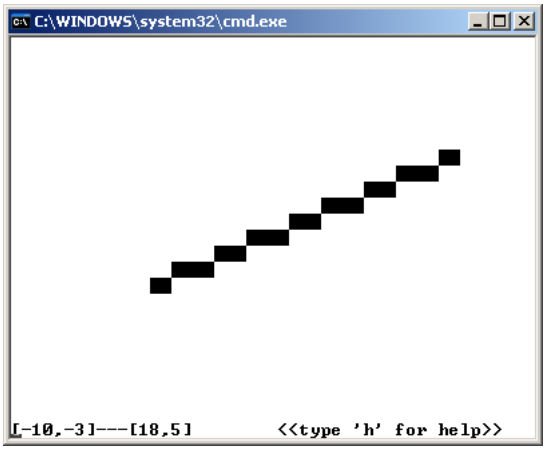

Abb. 1.1

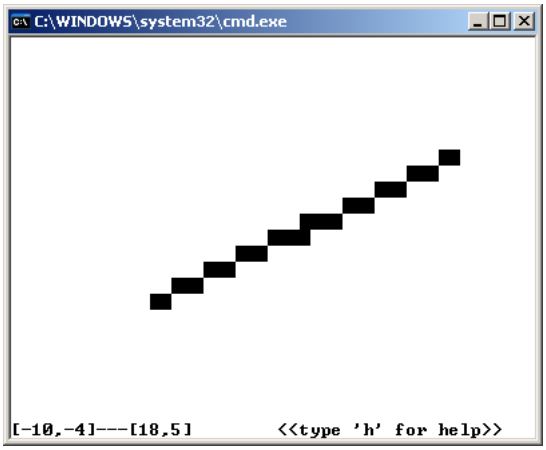

Abb. 1.2

Nach eingehender Betrachtung der Bilder finden Sie heraus, wie sie erzeugt wurden; es handelt sich nämlich

X	um die nach Bresenham erzeugte Verbindung zweier Punkte, die in beide Richtungen gezogen wurde.
	um zwei Linien, die nach Bresenham von benachbarten Startpunkten zu ebenfalls benachbarten Zielpunkten gezogen wurden.
	um zwei Linien, die benachbarte Start- und Zielpunkte verbinden und sich kreuzen.
	um zwei Bilder einer ganz normalen Bresenham-Implementierung.

2. **Aufgabe** (45 Punkte)

Sie wollen das Spiel „Schiffeversenken“ mit Grafik untermalen (Abb. 2.1) und grafisch darstellen, wie das um die y-Achse (als Linie entlang der z-Achse) modellierte Radar der Länge **2L** am Schiffsmast (Höhe **H**) sich um 90° dreht (Winkel **α**), während das Schiff von einem Torpedo getroffen wird, noch eine kurze Strecke **S** entlang der x-Achse fährt und dann um 90° (Winkel **β**) kippt, wodurch das Radar auf der Wasseroberfläche (y=0) zum Liegen kommt.

Abb. 2.1

Sie wollen die dazugehörige Gesamttransformation für das Radar in folgenden vier Schritten berechnen:

(i) Verschiebung entlang der y-Achse vom Koordinaten-Ursprung bis zur Höhe **H** > 0

(ii) Drehung um die y-Achse (Winkel **α**), bis das eine Radar-Ende, das über der negativen z-Achse lag, über die positive x-Achse kommt

(iii) Verschiebung entlang der x-Achse um die Strecke **S** > 0 und

(iv) Drehung um die x-Achse (Winkel **β**), bis sämtliche Ausdehnungen in positiver y-Richtung in ebensolche in positiver z-Richtung übergehen.

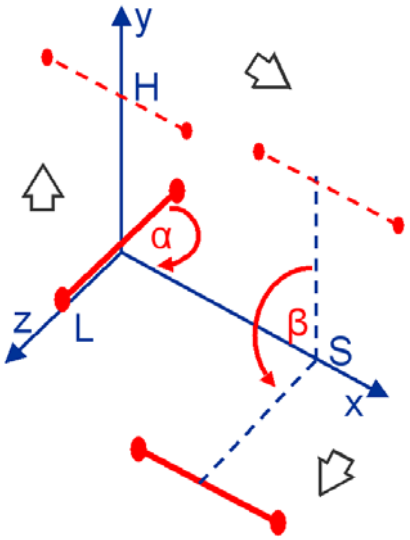

Abb. 2.2

Lösen Sie bitte diese Aufgabe, indem Sie nacheinander folgende Fragen bearbeiten:

a) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(i)}$, mit welcher das Radar auf die Höhe **H** des Mastes über den Koordinaten-Ursprung verschoben wird?

$$\underline{\mathbf{T}}_{(i)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) Geben Sie (in Grad, unter Berücksichtigung des Drehsinns) den Winkel **α** an, um welchen das Radar gedreht wird, damit seine über der negativen z-Achse liegende Hälfte über die positive x-Achse kommt. Geben Sie bitte auch **sin α** und **cos α** an!

α = *– 90°* **sin α =** *–1* **cos α =** *0*

c) Wie lautet die Transformationsmatrix $\underline{T}_{(ii)}$, die das Radar wie unter b) beschrieben um die y-Achse dreht und parallel zur x-Achse stellt? Geben Sie sie bitte sowohl in symbolischer (sin α, ...) als auch in arithmetischer (zahlenmäßiger) Form an!

$$\underline{T}_{(ii)} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

d) Wie lautet die Transformationsmatrix $\underline{T}_{(iii)}$, die das Radar entlang der x-Achse um die Länge **S** verschiebt?

$$\underline{T}_{(iii)} = \begin{pmatrix} 1 & 0 & 0 & S \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

e) Geben Sie nun bitte (in Grad, unter Berücksichtigung des Drehsinns) den Winkel **β** an, um welchen das Radar um die x-Achse gedreht wird, damit es auf die o.a. Endstellung kommt. Geben Sie bitte auch **sin β** und **cos β** an!

β = *90°* **sin β =** *1* **cos β =** *0*

f) Wie lautet die Transformationsmatrix $\underline{T}_{(iv)}$, die das Radar wie unter e) beschrieben um die x-Achse dreht? Geben Sie bitte auch hier sowohl die symbolische als auch die arithmetische Form an!

$$\underline{T}_{(iv)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

g) Wie berechnet sich die gesuchte Transformationsmatrix $\underline{T}_{gesamt}$ für die Koordinaten der Radar-Enden aus den bisher besprochenen Transformationen $\underline{T}_{(i)}$ bis $\underline{T}_{(iv)}$?

$$\underline{T}_{gesamt} = \underline{T}_{(iv)} \bullet \underline{T}_{(iii)} \bullet \underline{T}_{(ii)} \bullet \underline{T}_{(i)}$$

h) Berechnen Sie jetzt bitte die gesuchte Transformationsmatrix $\underline{T}_{gesamt}$ nach den obigen Angaben.

$$\underline{T}_{gesamt} = \underline{T}_{(iv)} \bullet [\, \underline{T}_{(iii)} \bullet [\, \underline{T}_{(ii)} \bullet \underline{T}_{(i)} \,] \,] \,]$$

$\underline{T}_{(iii)}$

$$\begin{pmatrix} 1 & 0 & 0 & S \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(ii)}$

$$\begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(i)}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(iv)}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(iv)} \bullet \underline{T}_{(iii)}$

$$\begin{pmatrix} 1 & 0 & 0 & S \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(iv)} \bullet \underline{T}_{(iii)} \bullet \underline{T}_{(ii)}$

$$\begin{pmatrix} 0 & 0 & -1 & S \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{gesamt}$

$$\begin{pmatrix} 0 & 0 & -1 & S \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

i) Berechnen Sie nun bitte auch die Koordinaten der Radar-Enden in ihrer Endstellung. (Eventuelle Folgefehler werden als richtig angerechnet.)

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -L & L \\ 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & -1 & S \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} S-L & S+L \\ 0 & 0 \\ H & H \\ 1 & 1 \end{pmatrix}$$

j) Ein Bekannter fragt Sie, ob sich an der Berechnung der Endposition etwas ändern würde, wenn im Szenario der Versenkung das Radar sich erst nach der kurzen Fahrt (Strecke **S**) entlang der x-Achse drehen würde.

Was antworten Sie darauf?

Am vorgestellten Rechenweg würde sich nichts ändern: Das Radar muß um seine Modellierungsachse drehen, sonst verläßt es das Schiff, indem es einen Bogen mit Radius S um die y-Achse beschreibt. Bei einer Animation müßte es evtl. zurückverschoben, gedreht und wieder vorgeschoben werden.

3. **Aufgabe** (45 Punkte)

Ermutigt durch Ihre Erfolge in der Computergrafik begannen Sie vor langer Zeit, Ihre eigene OpenGL-Version des Spiels „Schiffeversenken“ aus der U-Boot-Perspektive zu entwickeln (Abb. 3.1); das erhöhte wiederum Ihren Bekanntheitsgrad, so daß Sie inzwischen für andere Grafik-Projekte abgezogen wurden.

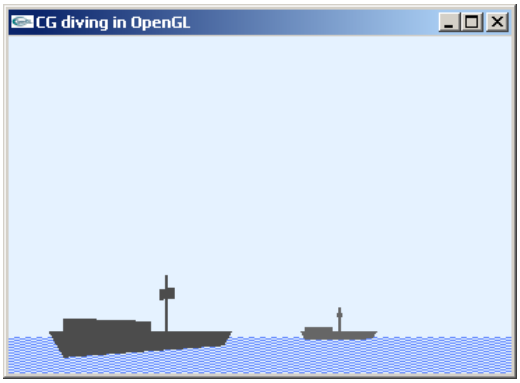

Abb. 3.1

Nach langer Pause erklären Sie nun das von Ihnen einmal erstellte Programm (bestehend aus den Dateien `periscope.c`, `periscope.h`, am Ende dieser Unterlage) interessierten Kollegen, die Ihr Werk fortsetzen wollen. Die von Ihnen gewissenhaft eingehaltene Kommentierung hilft Ihnen, sich im alten Code zu orientieren.

Beantworten Sie bitte folgende Fragen:

a) Das GLUT-Fenster wird in `main()` eingerichtet und besteht aus 360x240 Pixeln. Ihre Nachfolger wollen dies in ein quadratisches Fenster von 200x200 Pixeln umwandeln. Welche Anweisung muß dazu umgestellt werden? (Bitte die geänderte Fassung angeben!)

```
glutInitWindowSize(200, 200);
```

b) Zwei weitere Anweisungen in `main()` enthalten Angaben über 360 bzw. 240 Pixel. Welche Fenster-Einstellung(en) betreffen sie, und worin unterscheidet sich ihre Wirkung?

`glutInitWindowPosition(360, 240);` setzt die Voreinstellung für die Position aller vom Programm geöffneten Fenstern bei den Koordinaten (360, 240).

`glutPositionWindow(360, 240);` positioniert das aktuelle Fenster bei (360, 240).

c) Die Funktion `init()` setzt die Löschfarbe und die „Konstruktionsdaten" der (untereinander gleichen) Schiffe nach Abb. 3.1, die alle während des Programmlaufs unverändert bleiben. Durch Verwendung der Konstanten `BODY_X` und `BODY_Y` (aus `periscope.h`), die auch in die globalen Variablen `nah` und `fern` eingehen, wird u.a. sichergestellt, daß bei Veränderung der Fenstergröße die Relationen gewahrt bleiben. Bei der Betrachtung der Radar-Abmessungen (doppelt indizierte Variable `radar[][]`) erkennen Sie, welche geometrische Form es noch in `init()` hat, nämlich (Zutreffendes bitte markieren):

Quadrat	Quader	Raute	Rechteck	Würfel

d) Die Schiffsfigur wird in der Funktion `ship()` anhand der Abmessungen in `init()` gezeichnet. Bei näherer Betrachtung fällt auf, daß `init()` keine Angaben über die Dimensionen des Mastes enthält (wo das Radar angebracht ist).

Wie ist der Mast realisiert, und worin unterscheidet sich seine perspektivische Darstellung in Höhe, Breite und Tiefe bei nahen und entfernten Darstellungen gegenüber einem kreisrunden, entsprechend schmalen und hohen Zylinder?

Der Mast ist als Strich realisiert. Das bedeutet, daß seine Höhe perspektivisch korrekt dargestellt wird, seine Dicke (Breite und Tiefe) aber, anders als bei einem modellierten Zylinder, immer gleich bleibt.

e) Einem OpenGL-Neuling, der danach fragt, erklären Sie, daß die Koordinaten-Transformationen, die in `ship()` auf das Radar angewandt werden, in der Reihenfolge ihres Aufrufs bedeuten:

(a) Verschiebung um ¾ der Mastlänge in y-Richtung (Transformation $\underline{\mathbf{T}}_T$)

(b) Rotation um 90° um die x-Achse (Transformation $\underline{\mathbf{T}}_{R(x)}$)

(c) Rotation um 90° um die y-Achse (Transformation $\underline{\mathbf{T}}_{R(y)}$)

(d) Skalierung mit Faktor 0,3 entlang der x-, mit 0,6 entlang der y-Achse (Transformation $\underline{\mathbf{T}}_S$). (b.w.) ⇒

Die daraus resultierende Gesamttransformation kann der mathematisch versierte Kollege auch von Hand ausrechnen, als Ergebnis des folgenen Matrizenprodukts (bitte angeben):

$$\underline{T}_{gesamt} = \underline{T}_T \bullet \underline{T}_{R(x)} \bullet \underline{T}_{R(y)} \bullet \underline{T}_S$$

f) Wie verändert sich die Programm-Grafik von Abb. 3.1, wenn der Aufruf `glScalef()` von unterster an oberste Stelle der vier Transformationen gesetzt wird?

Bitte kreuzen Sie das dazugehörige Bild an:

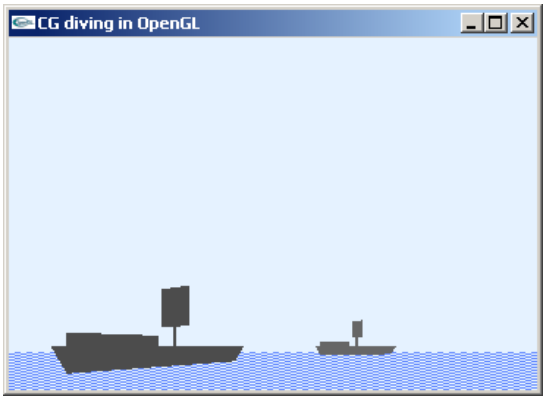

Abb. 3.2a Bitte hier ankreuzen: **( )**

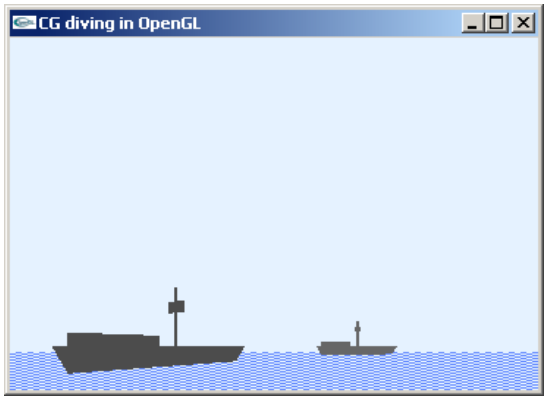

Abb. 3.2b Bitte hier ankreuzen: **( )**

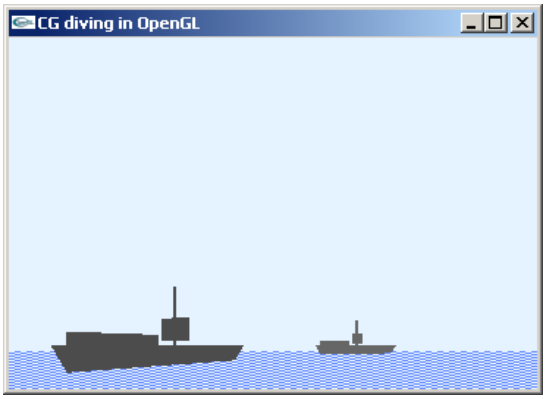

Abb. 3.2c Bitte hier ankreuzen: **(*X*)**

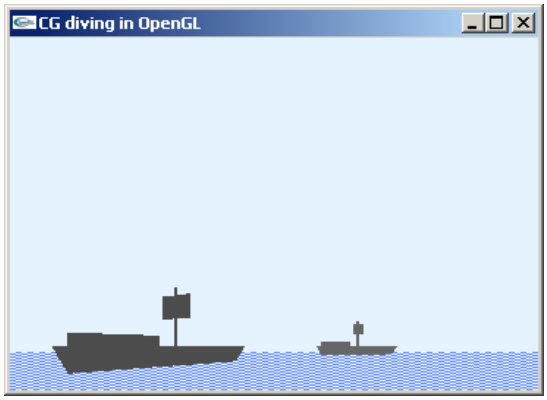

Abb. 3.2d Bitte hier ankreuzen: **( )**

Erklären Sie bitte stichwortartig, was Sie veranlaßt hat, sich für eines der obigen Bilder zu entscheiden (ein Argument genügt):

Die Skalierung, die ursprünglich (als letzter OpenGL-Aufruf) am Anfang der Transformationsfolge stand und sich nur auf das Radar auswirkte, betrifft nun auch die Radar-Positionierung.

(Aber auch die Skalierungsfaktoren treffen andere Dimensionen des Radars – daher die Größenänderung.)

g) Die Funktion `seascape()` zeichnet das Meer als gemusterte Fläche. Welche indizierten Variablen geben die Farben an, die sich im Muster abwechseln?

Welche der beiden Variablen stellt die hellere Farbe dar, und woran erkennen Sie das?

`water[3]={.4f, .55f, 1.f}`

`clear[3]={.9f, .95f, 1.f}`

`clear[]` hat für alle Komponenten (R/G/B) die höheren Werte und ist somit die hellere (Himmel-) Farbe.

h) Auf Wunsch Ihrer Freunde erweitern Sie den Code von `periscope` um ein paar Zeilen so, daß das Muster das ganze Fenster füllt (Abb. 3.3). Man fragt Sie, ob es nicht ein Merkmal (ein Pixel oder eine Pixel-Sequenz) innerhalb des Fensters gibt, woran man überprüfen kann, ob das gewünschte Muster richtig angewandt wurde.

Gibt es eine Stelle, die für ein gegebenes Muster unabhängig von der Fenster-Ausdehnung immer gleich bleibt?

Wenn ja: Welche Stelle ist dies, und wie ist sie beim aktuellen Muster besetzt? (Begründung!)

Wenn nein: Warum kann es eine solche Stelle nicht geben? (Begründung!)

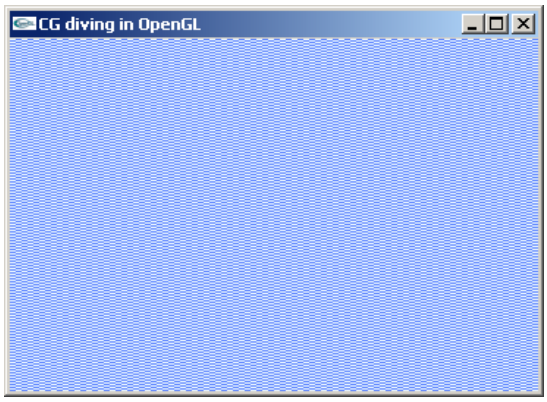

Abb. 3.3

Ja, die Stelle gibt es: Sie ist die untere linke Ecke des Fensters, wo die Belegung mit dem Muster beginnt. Hier geht es um das Muster `stip[]`, das mit $0xF0_{16}$ = $1111\ 0000_2$ beginnt („1“ steht für „Setzen“). Das heißt, unabhängig von der jeweiligen Ausdehnung des Fensters, die untere linke Ecke hat immer die Muster-Farbe – hier: `water[]`. Für das obige Bild bedeutet das, daß das Pixel am linken Ende der untersten Zeile immer ein dunkles ist.

i) Die Organisation der gesamten Szenerie erfolgt in der Funktion `draw()`. Dort werden auch die beiden Schiffe an unterschiedliche Positionen im Bild gesetzt. Dazu verwendet das Programm das Anweisungspaar `glPushMatrix()` und `glPopMatrix()`. Seine Wirkungsweise wird oft plakativ mit den Sätzen: „Merke Dir!“ und Erinnere Dich!“ umschrieben.

Bitte erläutern Sie kurz, was und zu welchem Zweck hier das Programm `periscope` sich merken soll, um es wieder abzurufen.

Mit der Anweisung `glPushMatrix()` „merkt sich“ das Programm die Positionierung der Gesamtszenerie (s. Code) und fügt jene des jeweiligen Schiffes hinzu. Mit `glPopMatrix()` verwirft das Programm wieder die hinzugefügte Transformation, so daß zur übrig bleibenden, „gemerkten“ Transformation der Gesamtszenerie, die Transformationen weiterer Objekte (z.B. des zweiten Schiffs) hinzugefügt werden können.

j) Die Positionierung des Augenpunkts erfolgt in der Funktion `positEye()`. Dort ist das Sichtvolumen zwar in x-Richtung, nicht aber in y-Richtung symmetrisch um die z-Achse angelegt. Der Versuch, eine Symmetrie auch in y-Richtung einzuhalten, erzeugt eine inakzeptable Szene (Abb. 3.4) mit einem Rand in der Löschfarbe am unteren Ende des Bildes. Dieser Rand läßt sich nur durch einen höher gelegten Augenpunkt vermeiden (Abb. 3.5). Eine solche Maßnahme ist aber mit der U-Boot-Perspektive des Programms unvereinbar.

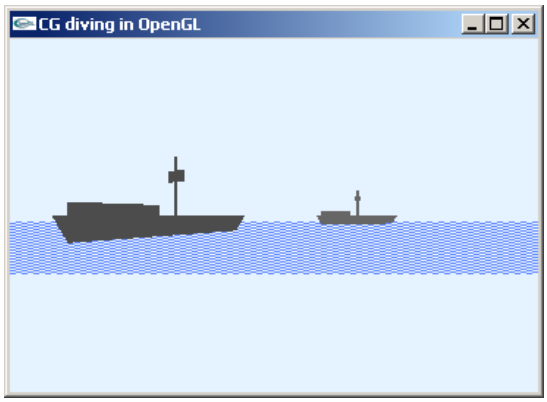

Abb. 3.4

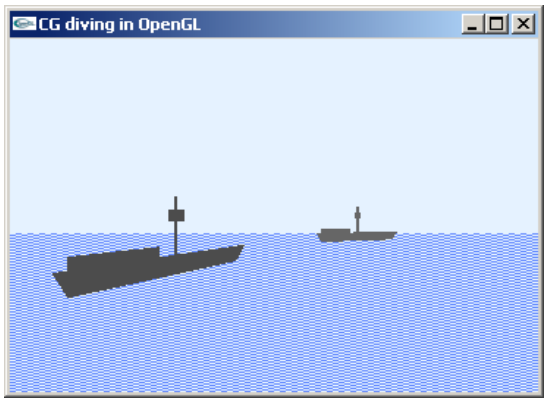

Abb. 3.5

Beantworten Sie bitte folgende Fragen:

Wie lautet die Anweisung, die (unter Berücksichtigung der Fenster-Abmessungen) ein Sichtvolumen anlegt, das in x- und y-Richtung symmetrisch um die z-Achse liegt?

`glFrustum(-180*FOV,180*FOV, -120*FOV, 120*FOV, nah,fern);`

Wie erklären Sie sich (intuitiv, ohne Rechnung) den unteren Rand in Abb. 3.4 und seinen Wegfall in Abb. 3.5 nach Höhersetzen des Augenpunktes (durch Änderung der y-Koordinate im Aufruf `glTranslatef()`)?

Der Effekt liegt an den near-Angaben in `glFrustum()`: Der untere Bild-Rand gehört zu Teilen der Szene, die näher als der Wert der Variablen nah liegen; deswegen wird an dieser Stelle das dort befindliche (Meer-) Objekt nicht gezeichnet, und es erscheint weiterhin die Himmel-Farbe, mit der zuvor das Fenster gelöscht wurde.

k) Jemand fragt Sie, ob es leichter gewesen wäre, das unsymmetrisch angelegte Sichtvolumen mit Anweisungen der GLU-Bibliothek anzulegen (z.B. mit `gluPerspective()`). Sie anworten, ohne es am Code auszuprobieren.

Wenn ja: Wie erleichtert die GLU-Library Ihre Arbeit an dieser Stelle?

Wenn nein: Warum ist GLU für diese Aufgabe ungeeignet?

Nein: GLU-Anweisungen sind hier ungeeignet, weil sie ein Sichtvolumen erzeugen, das symmetrisch um die eigene Mittelachse liegt.

```
/* periscope.c */
/*Darstellung eines U-Boot-Periskops mit OpenGL*/
#include "periscope.h"

/*Globale Variablen: */
float    tx=0., tz=0., ry=0.,
         clear[3]={.9f, .95f, 1.f}, water[3]={.4f, .55f, 1.f},
         steel[2][3]={{.4f, .4f, .4f}, {.3f, .3f, .3f}};
GLdouble nah=BODY_X, fern=10*BODY_X;
GLfloat  ocean[4][3], body[4][3], bridge[4][3], radar[4][3];

/*************************************************************************/
  void positEye(void)
/*************************************************************************/
/*Augenpunkt positionieren:*/
{ glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  /*Festlegung des Sichtvolumens:*/
  glFrustum (-180*FOV, 180*FOV, -30*FOV, 210*FOV, nah, fern);

  /*Aktualisierung der Blickrichtung:*/
  glRotatef(-ry, 0., 1., 0.);

  /*Konstanter (y-)Tiefstand, aktualisierte (x-,z-)Position:*/
  glTranslatef(-tx, -BODY_Y, -tz);
}

/*************************************************************************/
  void draw(void)
/*************************************************************************/
/*Grafik erstellen:*/
{ int j1=0;
  glClear(GL_COLOR_BUFFER_BIT);

  /*Positionierung Augenpunkt:*/
  positEye();

  /*Positionierung Gesamtszenerie:*/
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0., 0., -(nah+BODY_X));

  /*Zeichnung Ozean:*/
  seascape();

  /*Zeichnung Schiffe:*/
  for (j1=0; j1<2; j1++)
  { glPushMatrix();
    if (!j1) { glTranslatef(2*BODY_X, 0.f, -5*BODY_X);
               glRotatef(20, 0., 1., 0.); }
    else     { glTranslatef(-BODY_X, 0.f, -BODY_X);
               glRotatef(60, 0., 1., 0.); }
    glColor3fv (steel[j1]);
    ship();
    glPopMatrix();
  }
  glFlush();
}
```

```
/*************************************************************************/
  void seascape(void)
/*************************************************************************/
/*Wasser-Oberflaeche zeichnen:*/
{ int j1=0;

  /*Ozean:*/
  glEnable (GL_POLYGON_STIPPLE);
  glPolygonStipple (stip);
  glColor3fv (water);

  glBegin(GL_POLYGON);
    glVertex3fv(ocean[0]);   glVertex3fv(ocean[1]);
    glVertex3fv(ocean[2]);   glVertex3fv(ocean[3]);
  glEnd();
  glDisable (GL_POLYGON_STIPPLE);
  return;
}

/*************************************************************************/
  void init(void)
/*************************************************************************/
/*Initialisierungen:*/
{
  /*Loeschfarbe:*/
  glClearColor (clear[0], clear[1], clear[2], 1.f);

  /*Eckpunkt-Koordinaten Ozean (gegen den UZS, beginnend v.li.):*/
  ocean[0][X] = ocean[1][X] = -fern;
  ocean[2][X] = ocean[3][X] =  fern;
  ocean[0][Y] = ocean[1][Y] = ocean[2][Y] = ocean[3][Y] = 0.f;
  ocean[0][Z] = ocean[3][Z] = -fern;
  ocean[1][Z] = ocean[2][Z] =  fern;

  /*Abmessungen Rumpf (gegen den UZS, beginnend o.li.):*/
  body[0][X] = -BODY_X*1.1f;  body[1][X] = -BODY_X;
  body[2][X] =  BODY_X;       body[3][X] =  BODY_X*1.2f;
  body[0][Y] =  body[3][Y] =  BODY_Y;
  body[1][Y] =  body[2][Y] =  0;
  body[0][Z] =  body[1][Z] =  body[2][Z] = body[3][Z] = 0.f;

  /*Abmessungen Bruecke (gegen den UZS, beginnend o.li.):*/
  bridge[0][X] = bridge[1][X] = -BODY_X;
  bridge[2][X] = bridge[3][X] = -BODY_X*.2f;
  bridge[0][Y] = bridge[3][Y] =  body[0][Y]*1.5;
  bridge[1][Y] = bridge[2][Y] =  body[0][Y];
  bridge[0][Z] = bridge[1][Z] = bridge[2][Z] = bridge[3][Z] = 0.f;

  /*Abmessungen Radar (gegen den UZS, beginnend o.li.):*/
  radar[0][X] = radar[1][X] = -BODY_Y;
  radar[2][X] = radar[3][X] =  BODY_Y;
  radar[0][Y] = radar[3][Y] = -BODY_Y;
  radar[1][Y] = radar[2][Y] =  BODY_Y;
  radar[0][Z] = radar[1][Z] = radar[2][Z] = radar[3][Z] = 0.f;

  /*Tastendruck simulieren, um Menue auszugeben:*/
  key(' ', 0, 0);
  return;
}
```

```
/*************************************************************************/
  void ship(void)
/*************************************************************************/
/*Ein Schiff zeichnen:*/
{ int j1=0;
  /*Rumpf:*/
  glBegin(GL_POLYGON);
    glVertex3fv(body[0]);   glVertex3fv(body[1]);
    glVertex3fv(body[2]);   glVertex3fv(body[3]);
  glEnd();
  /*Bruecke:*/
  glBegin(GL_POLYGON);
    glVertex3fv(bridge[0]);   glVertex3fv(bridge[1]);
    glVertex3fv(bridge[2]);   glVertex3fv(bridge[3]);
  glEnd();
  /*Mast:*/
  glLineWidth (2.5);
  glBegin(GL_LINES);
    glVertex3f(0.f,   BODY_Y, 0.f);
    glVertex3f(0.f, 4*BODY_Y, 0.f);
  glEnd();
  /*Radar:*/
  glTranslatef(0.f, 3*BODY_Y, 0.f);
  glRotatef(90.f, 1.f, 0.f, 0.f);
  glRotatef(90.f, 0.f, 1.f, 0.f);
  glScalef(.3f, .6f, 1.f);
  glBegin(GL_POLYGON);
    glVertex3fv(radar[0]);   glVertex3fv(radar[1]);
    glVertex3fv(radar[2]);   glVertex3fv(radar[3]);
  glEnd();
}
/*************************************************************************/
  int main(int argc, char **argv)
/*************************************************************************/
{ glutInitWindowPosition(360, 240);
  glutInitWindowSize(360, 240);
  glutInit(&argc, argv);
  glutCreateWindow("CG diving in OpenGL");
  glutPositionWindow(360, 240);
  glutDisplayFunc(draw);
  glutKeyboardFunc(key);
  init();
  glutMainLoop();
  return 0;
}
/*************************************************************************/
  void key(unsigned char key, int x, int y)
/*************************************************************************/
/*Menue und Eingabe-Behandlung:*/
{ const  double  GRAD  = atan(1.) / 45.;
  switch (key)
  { case ESC: exit(0);
    case 'a': ry += 5.;  if (ry >= 360) ry-=360; break;
    case 'd': ry -= 5.;  if (ry <=-360) ry+=360; break;
    case 's': tz += BODY_Y*cos(ry*GRAD); tx += BODY_Y*sin(ry*GRAD); break;
    case 'w': tz -= BODY_Y*cos(ry*GRAD); tx -= BODY_Y*sin(ry*GRAD); break;
  }  system (CON_CLS);
  printf ("\n\r Press (keeping the GLUT window activated):");
  printf ("\n\n\r <ESC>  to quit");
  printf ("\n\r a / d     turn 5 deg. l./r. (curr.%7.2f)", ry);
  printf ("\n\r s / w     go back / on (curr.%5.1f units)", -BODY_Y*tz);
  draw();   return; }
```

```c
/* periscope.h */

#ifndef PERISCOPE_H
  #define PERISCOPE_H
  #include <stdio.h> //wg. printf()
  #include <math.h>  //wg. sin, cos, atan
  #include <GL/glut.h>

  #define CON_CLS  "cls"
  #define ESC        27
  #define FOV          .025 //Field Of View
  #define BODY_X      5     //Schiffsrumpf
  #define BODY_Y      1

  enum {X=0, Y=1, Z=2, W=3};

  /*Prototypen:*/
  void draw(void);
  void init(void);
  void key(unsigned char key, int x, int y);
  void positEye(void);
  void seascape(void);
  void ship(void);

/*Globale Variable: */
  GLubyte stip[] = {
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F};
#endif //PERISCOPE_H
```

Platz für Notizen: